Performance auto-tuning of rectangular matrix-vector multiplication: how to outperform CUBLAS

Sørensen, Hans Henrik Brandenborg

Publication date:
2010

Document Version
Publisher's PDF, also known as Version of record

Performance auto-tuning of rectangular matrix-vector multiplication: how to outperform CUBLAS.

Hans Henrik Brandenborg Sørensen
Section for Scientific Computing
DTU Informatics

GPU-Lab
DTU Informatics

GPULab `http://gpulab.imm.dtu.dk`

New group at DTU Informatics 2010

- The current group members:

Per Christian Hansen	Allan Engsig-Karup	Stefan Lemvig Glimberg
Jan Hesthaven	Jeppe Frisvad	Nicolai Fog Gade-Nielsen
Bernd Dammann	Boyuan Lazarov	Toke Jansen Hansen
John Bagterp Jørgensen	Hans Henrik Brandenborg Sørensen	Morten Gorm Madsen

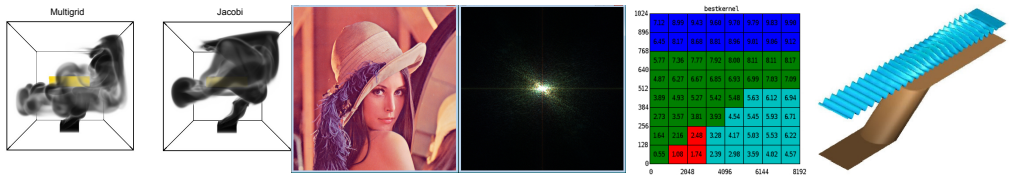
- We have many GPU activities, e.g.:

Ph.D. School in

Scientific GPU Computing

Copenhagen, May 2011

Projects: see webpage



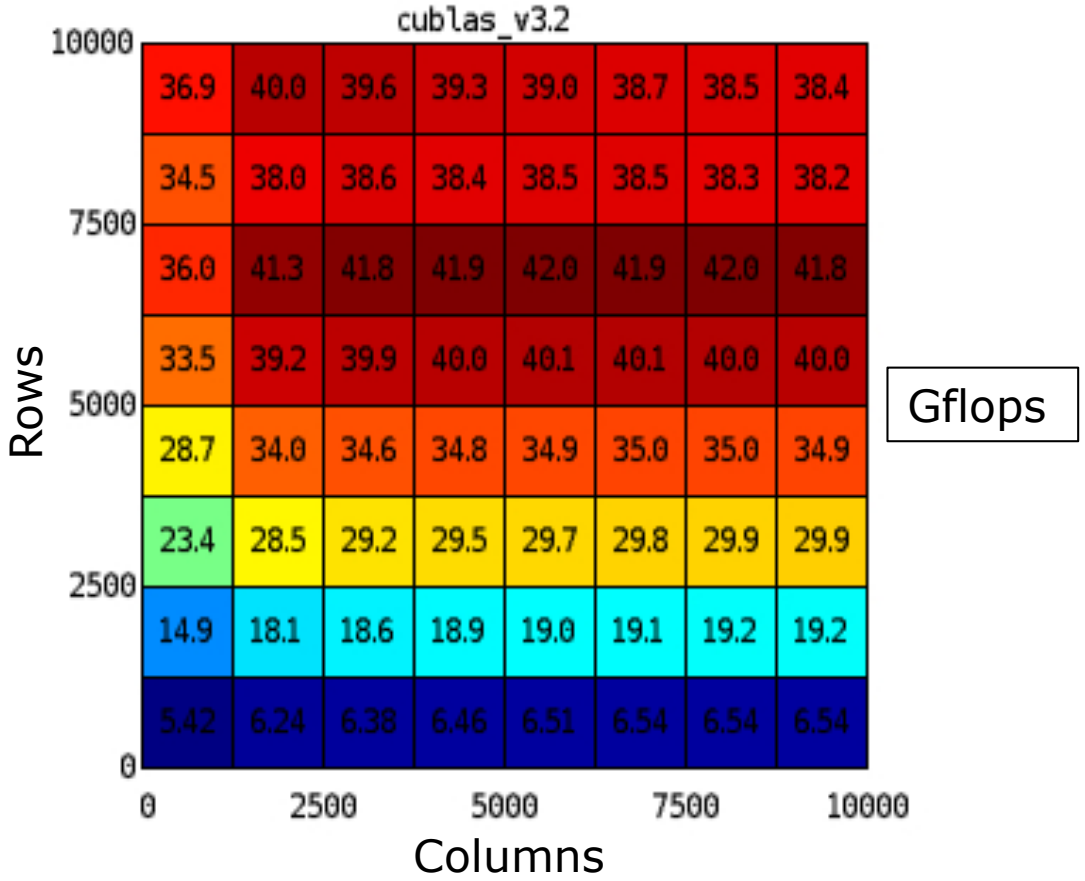
02614 High-performance computing

GPU-Lab
DTU Informatics

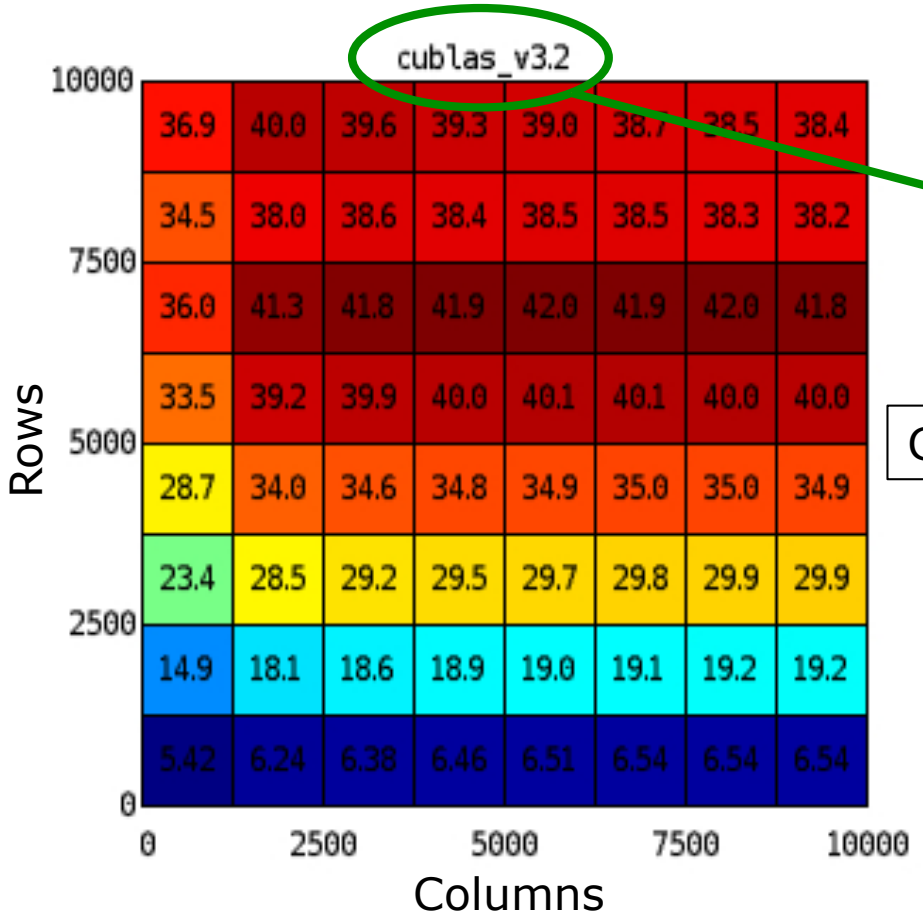
Outline

1. Why Auto-Tuning?
2. Example: A versatile matrix-vector kernel.
3. Results.

Timing Ax=y (Sgemv) on Tesla C2050



Timing Ax=y (Sgemv) on Tesla C2050



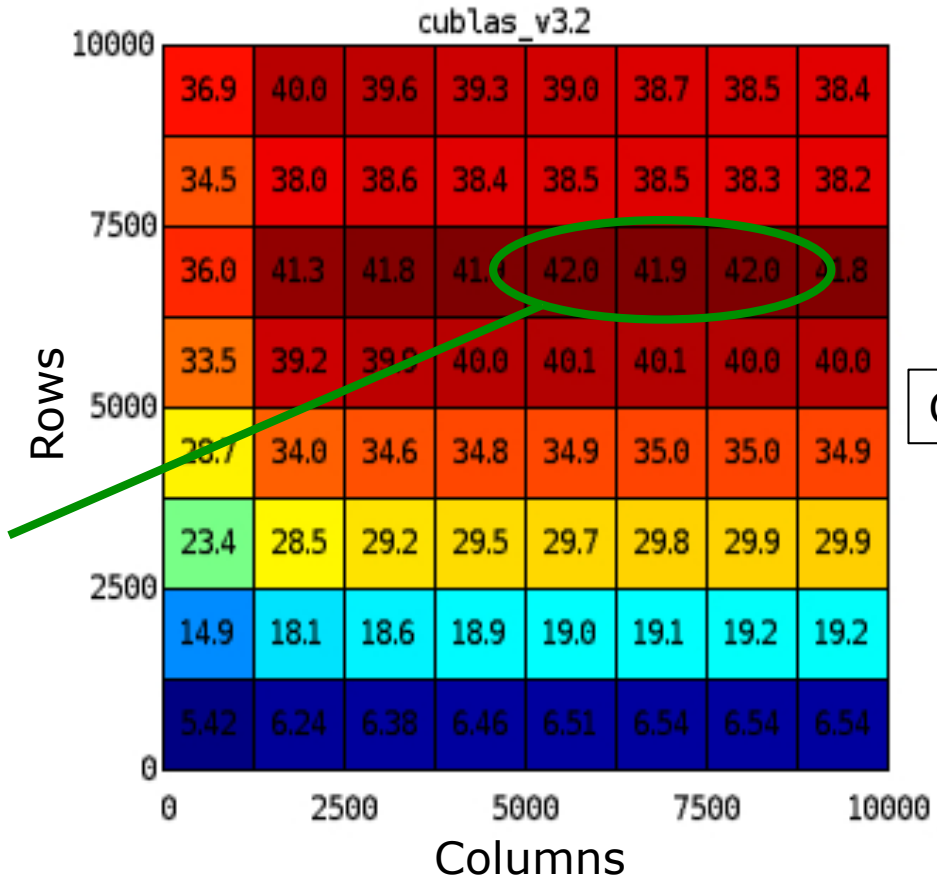
Nividia CUDA Basic Linear Algebra Subroutines.

Version 3.2.17
Released Nov. 2010

Gflops

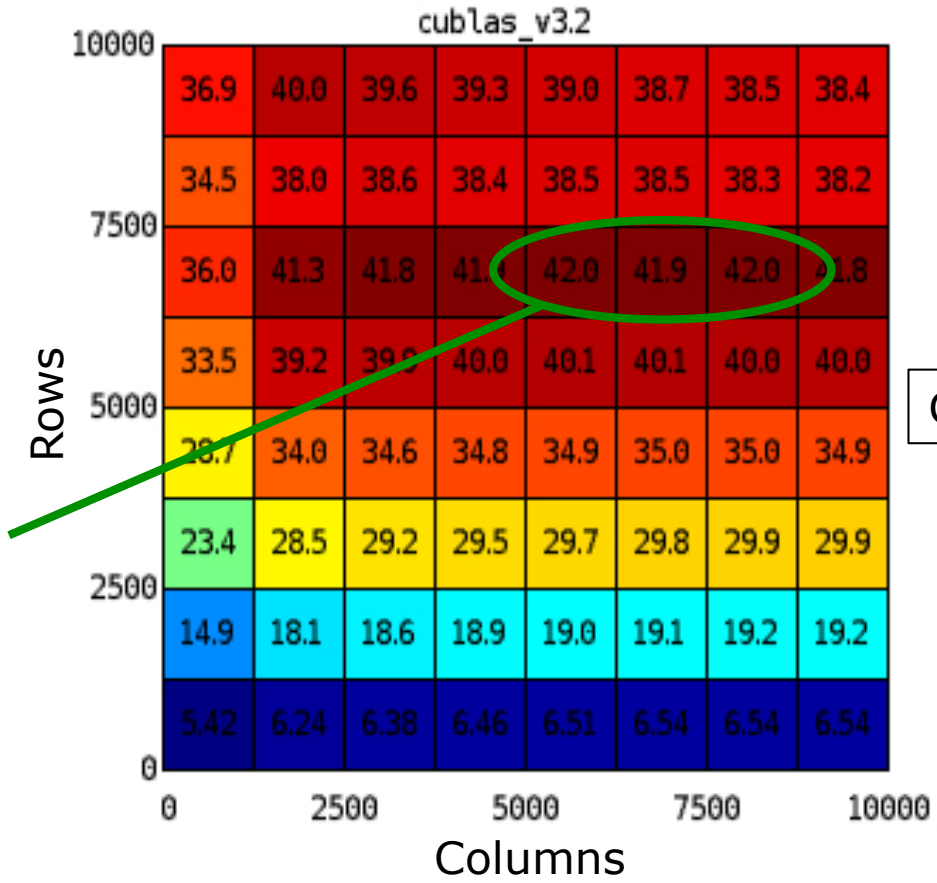
"CUDA BLAS (CUBLAS) library routines, delivering 8 times faster performance than the latest Intel MKL (Math Kernel Library) "

Timing Ax=y (Sgemv) on Tesla C2050



Best performance around 42 GfLops

Timing Ax=y (Sgemv) on Tesla C2050



Best performance around 42 GfLops

Tesla C2050:

Theoretical peak 1288 Gflops.

Theoretical bandwidth 144 GB/s.

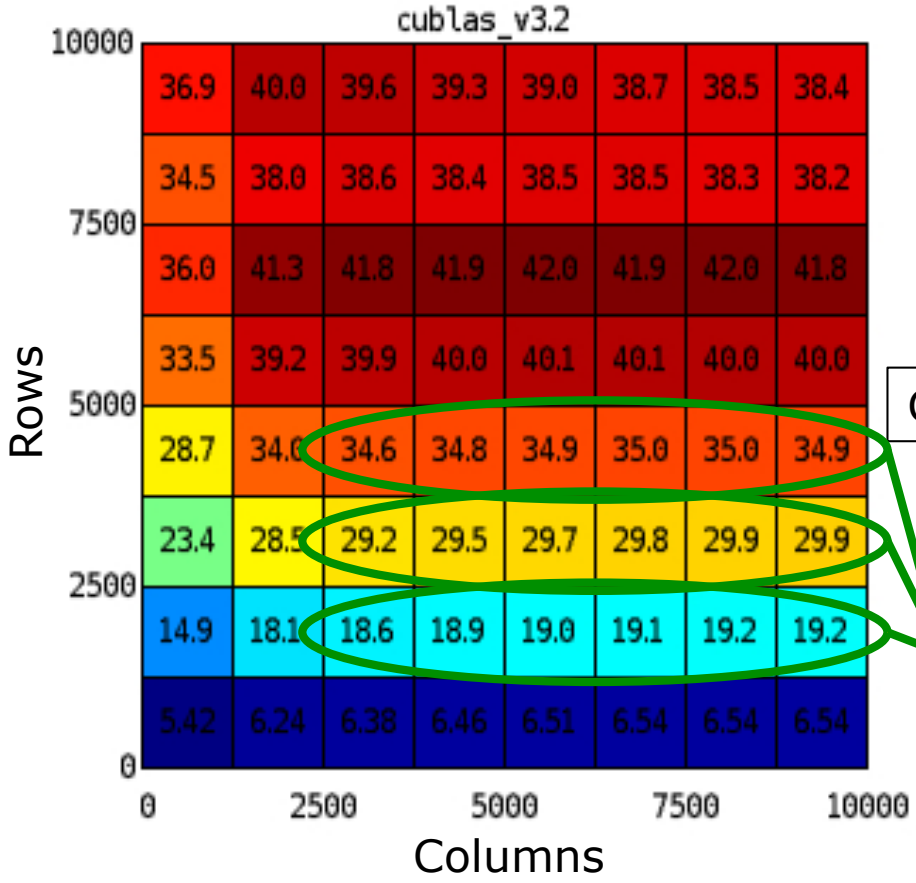
Effective (bandwidthTest) ~85 GB/s.

”Bandwidth bound”

Best performance: 85(GB/s) / 4bytes x 2flops = ~42.5 Gflops

GPU-Lab DTU Informatics

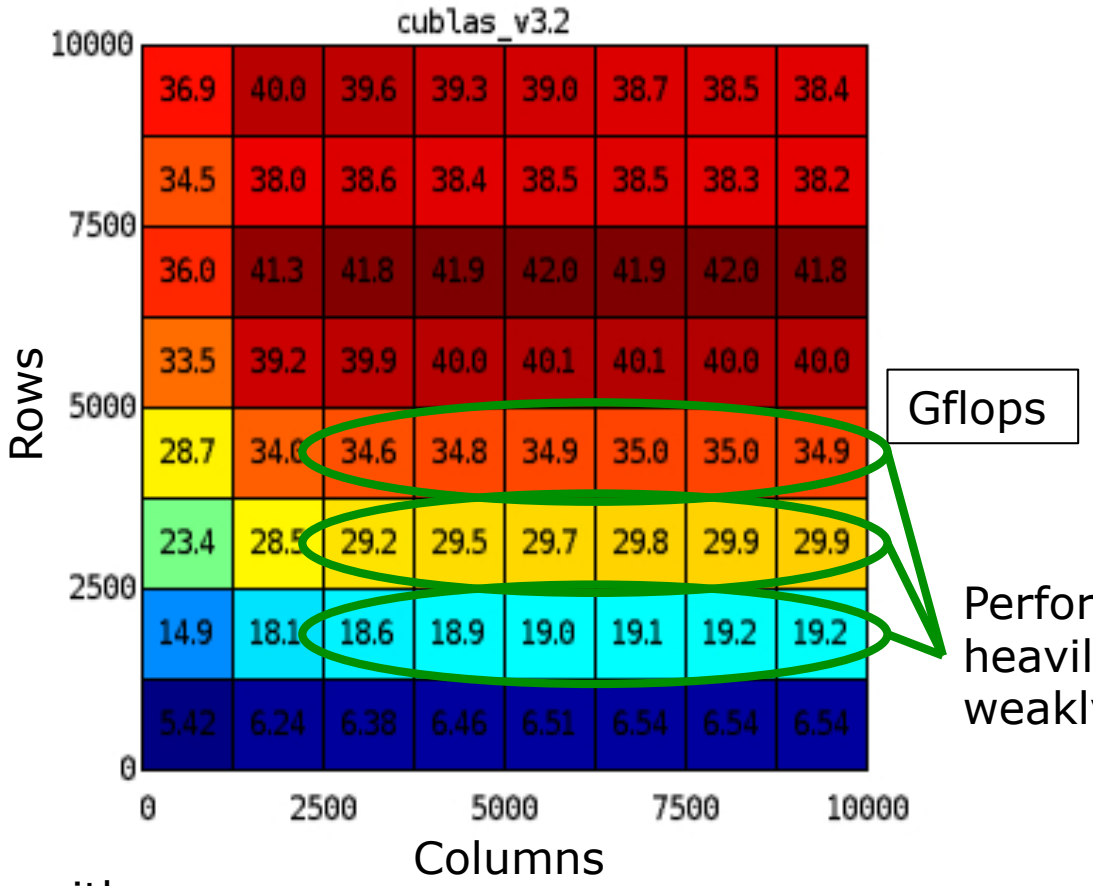
Timing Ax=y (Sgemv) on Tesla C2050



Performance depends heavily on rows, weakly on columns.

GPU-Lab
DTU Informatics

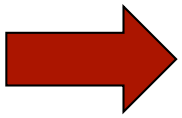
Timing Ax=y (Sgemv) on Tesla C2050



Performance depends heavily on rows, weakly on columns.

Typical Ax=y algorithm:

```
for i = 1:Rows
   for j = 1:Columns
      y(i) += A(i,j) * x(j);
   end
end
```

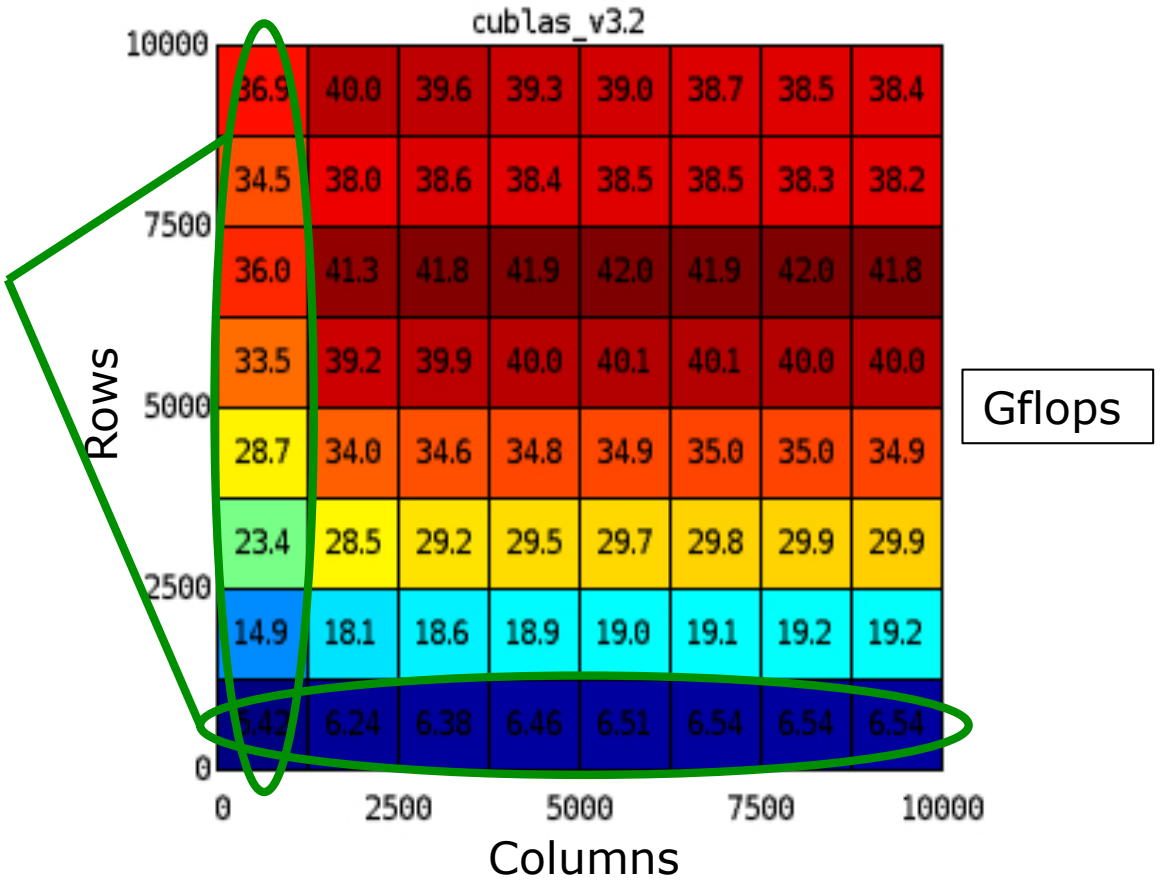


"One thread per row"-type

More rows improves performance until full occupancy.

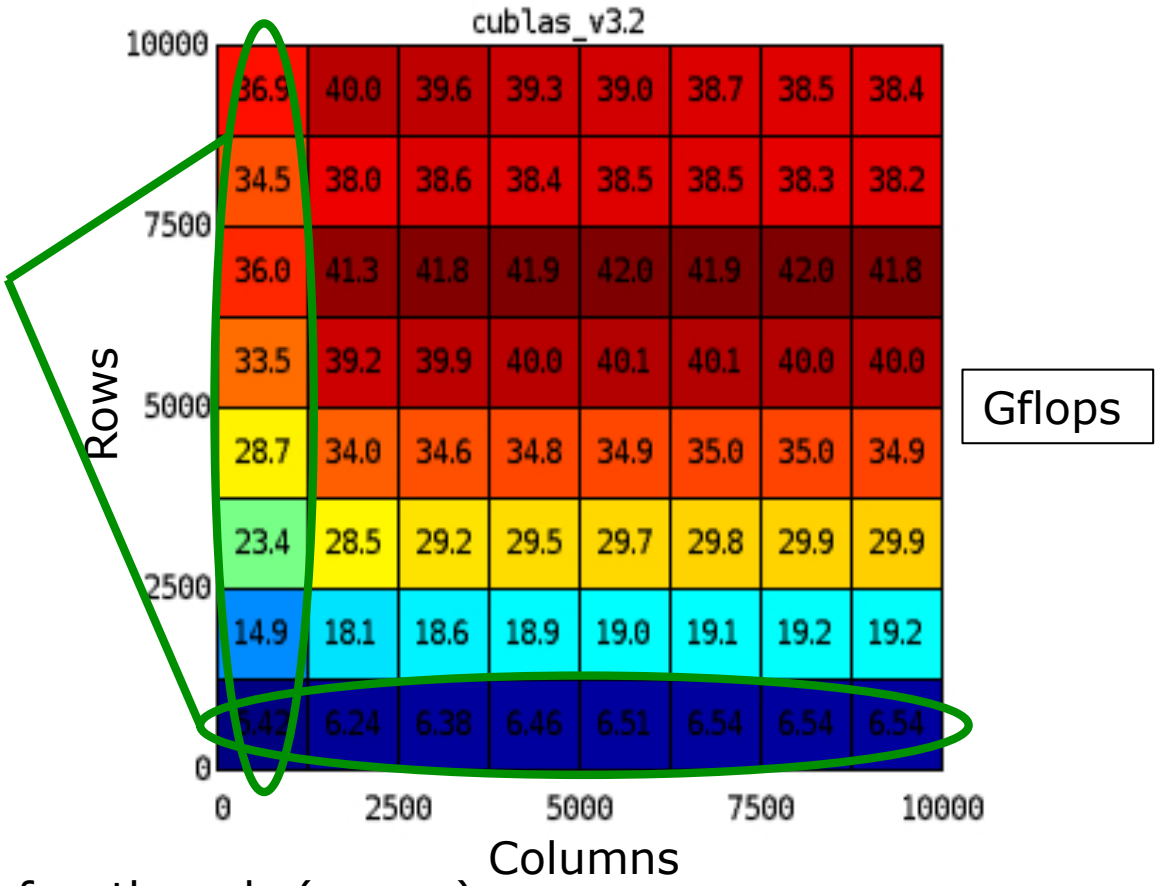
Timing Ax=y (Sgemv) on Tesla C2050

Rectangular matrix-vector performance is noticeable less than optimal.



Timing Ax=y (Sgemv) on Tesla C2050

Rectangular matrix-vector performance is noticeable less than optimal.



Wide A: Too few threads (=rows) to hide latency.

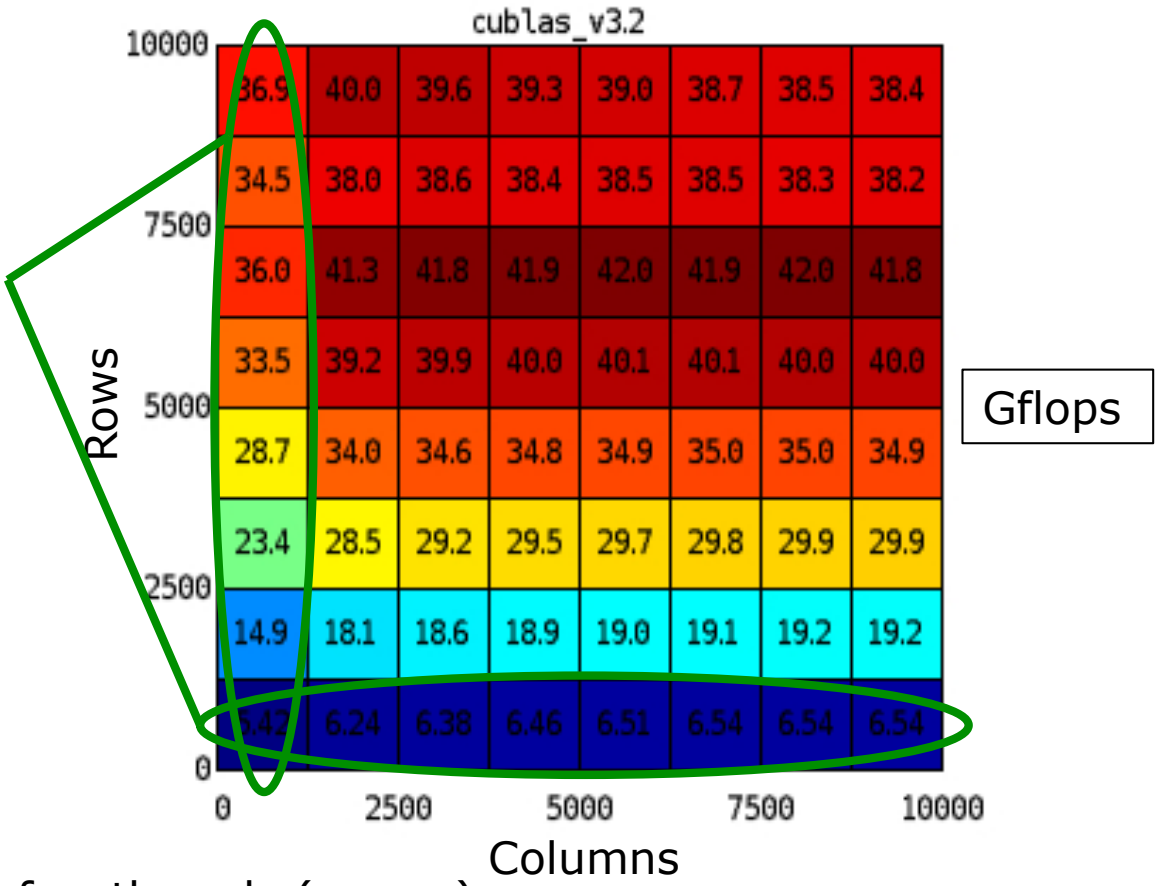
Tall A: Too little work (=columns) to hide overhead.

→ Implementation lacks versatility.

GPU-Lab DTU Informatics

Timing Ax=y (Sgemv) on Tesla C2050

Rectangular matrix-vector performance is noticeable less than optimal.

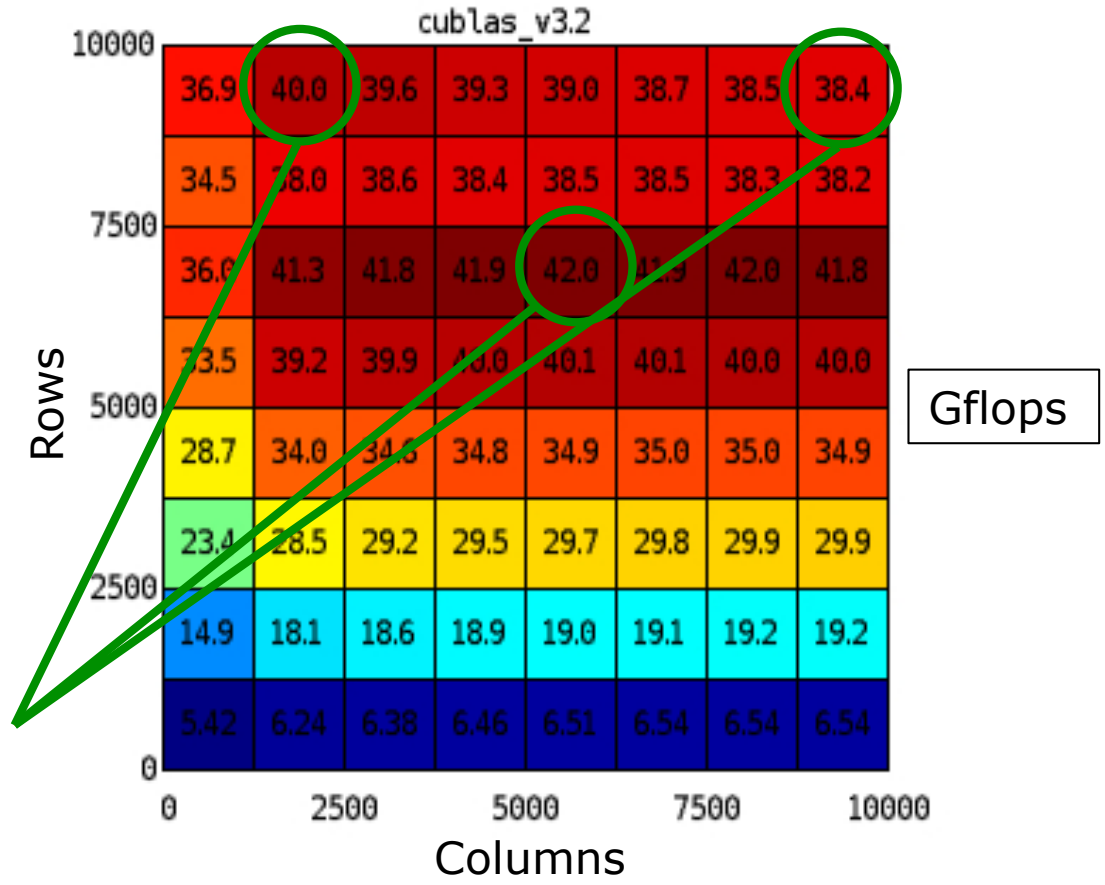


Wide A: Too few threads (=rows) to hide latency.

Tall A: Too little work (=columns) to hide overhead.

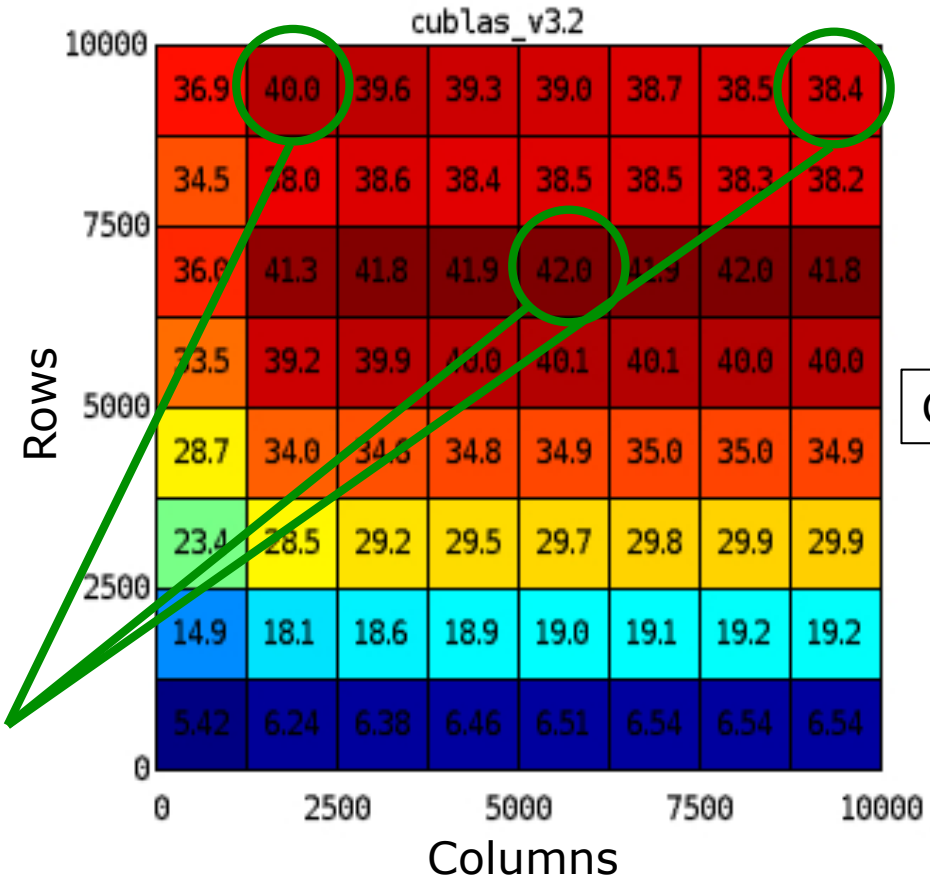
→ **C++ Templates** Implementation specific.

Timing Ax=y (Sgemv) on Tesla C2050



Performance differences due to CUDA and hardware.

Timing Ax=y (Sgemv) on Tesla C2050



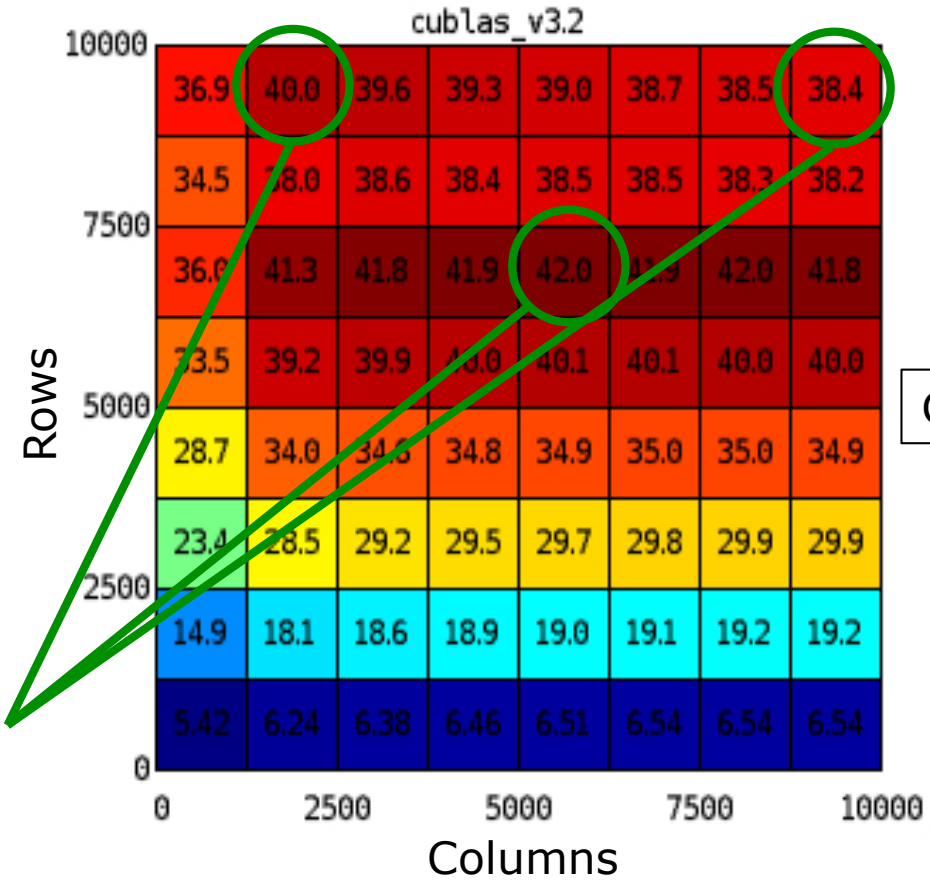
Performance differences due to CUDA and hardware.

Complicated interplay between gridsize, blocksize, padding of A, smem/register usage etc. and hardware specifications.

CUDA C Best Practices Guide 3.2: ” There are many such factors involved ... , and inevitably some experimentation is required.”

GPU-Lab
DTU Informatics

Timing Ax=y (Sgemv) on Tesla C2050



Performance differences due to CUDA and hardware.

Complicated interplay between gridsize, blocksize, padding of A, smem/register usage etc. and hardware specifications.

CUDA C Best Practices Guide 3.2: "There are many such factors involved, and their variability is the experimentation is required."

Auto-tuning

GPU-Lab
DTU Informatics

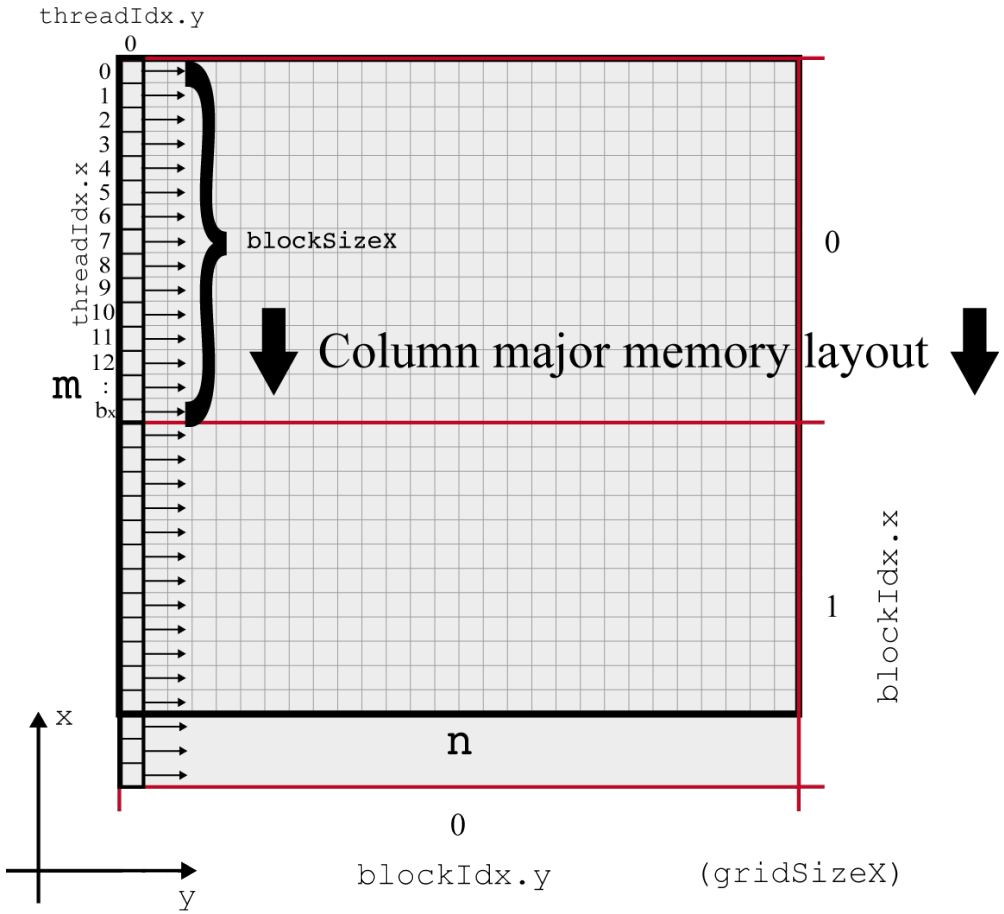
Outline

2. Example: A versatile matrix-vector kernel.

Simple matrix transversal kernel

Coalesced memory access

- Consecutive threads should access consecutive locations.
- Loops should progress perpendicular to major storage direction.



One thread per row

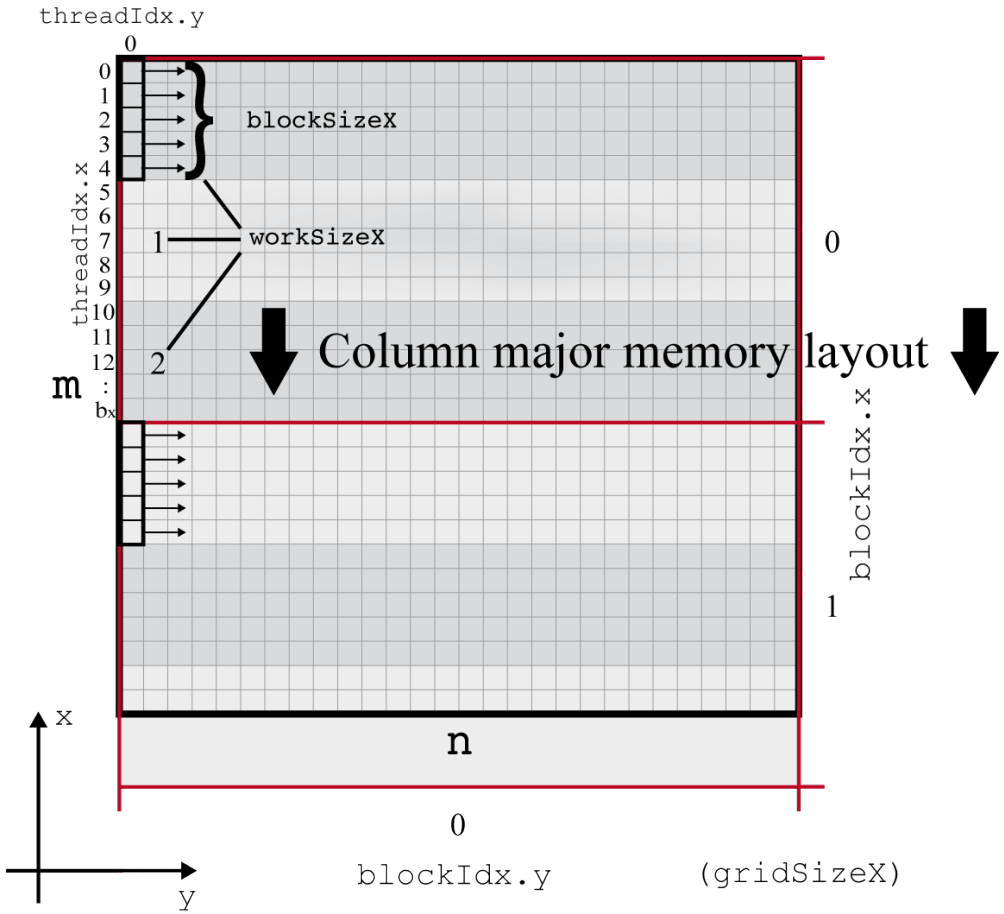
- m <=`blockSizeX*gridSizeX`.
- Performs some operation on each element in its row;
- +,x,• ➔ sum, norm, dot!

GPU-Lab
DTU Informatics

Better matrix transversal kernel

Threads that do more rows each

- For tall matrices; we avoid ”many threads that do very little work each”.
- This requires a new parameter: `workSizeX`.



Several rows per thread

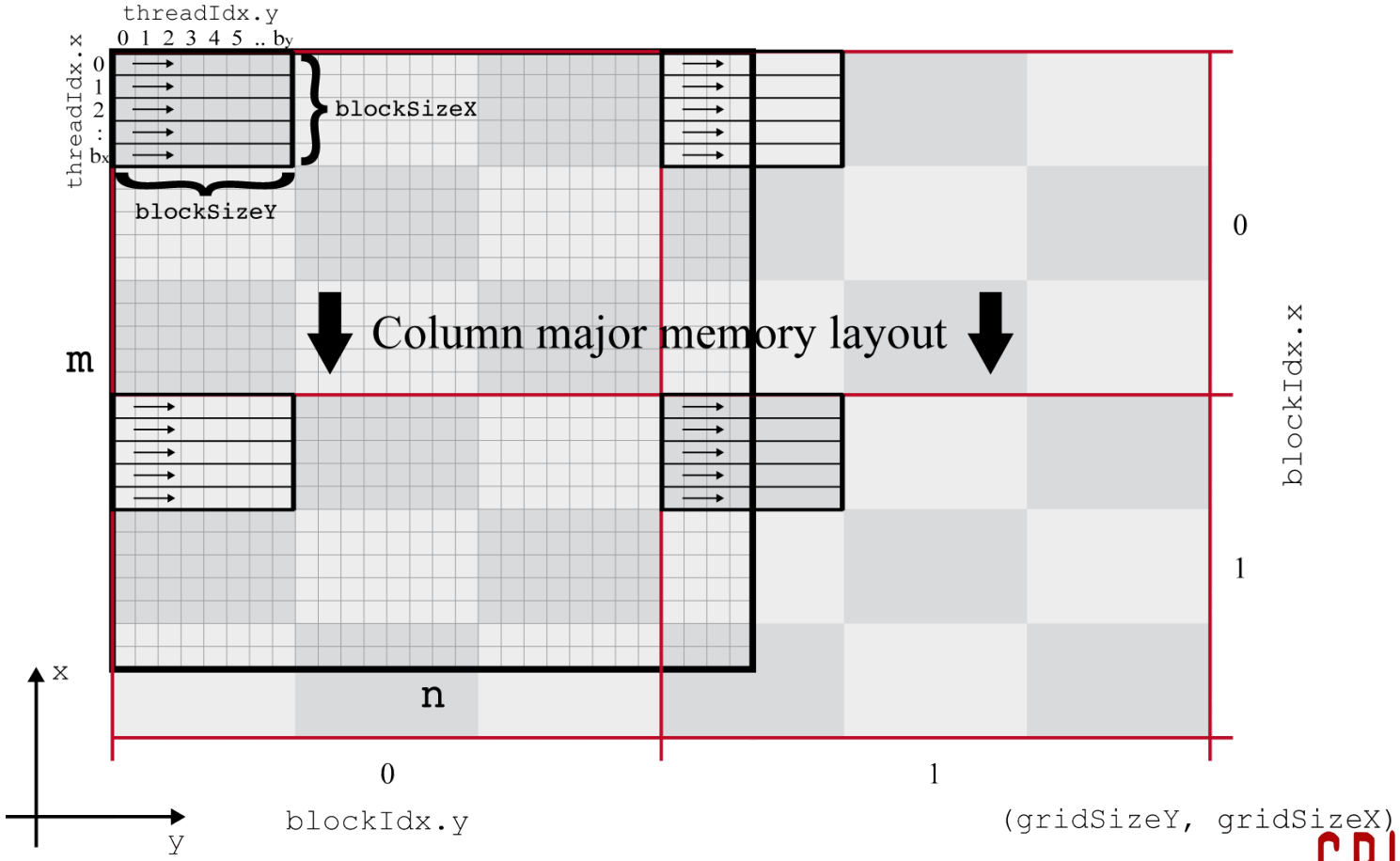
- m <= `workSizeX * blockSizeX * gridSizeX`.
- Performs some operation on each element in a row and then moves on to the next until its work is done.

GPU-Lab
DTU Informatics

Versatile transversal kernel

Threads work together on a row

- For wide matrices; we avoid ”very few threads that do all the work”.
- Parameters: `blockSizeX, blockSizeY, workSizeX, workSizeY`.

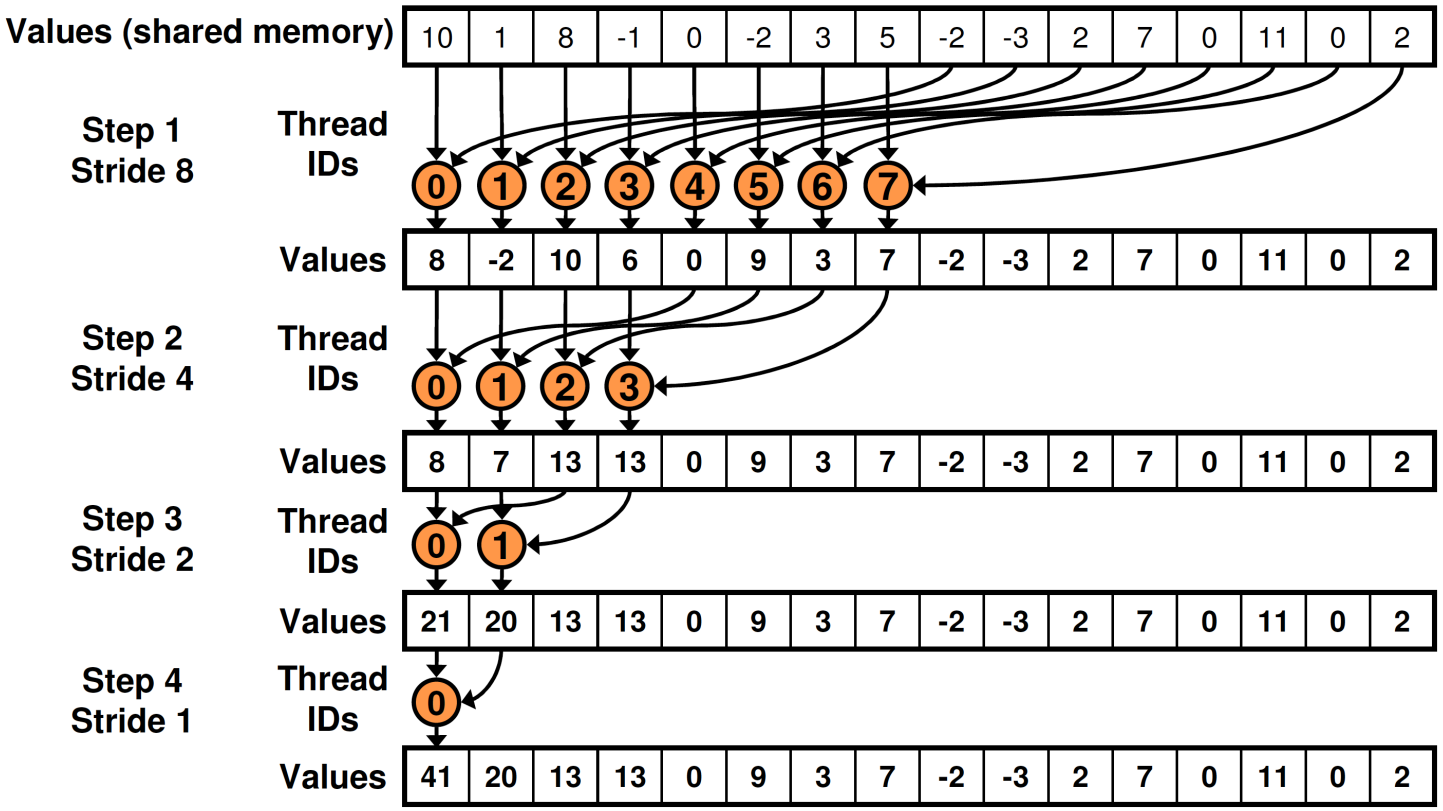


GPU-Lab
DTU Informatics

Parallel reduction

Threads working together on a row requires parallel reduction

- Mark Harris' approach (`http://developer.download.nvidia.com/compute/cuda/1_1/Website/projects/reduction/doc/reduction.pdf`)



How to template the kernel ?

CUDA supports C++ templates (some functionality)

- Template the sgemv kernel as:

```
template <int blockSizeX, int blockSizeY, int workSizeX, int workSizeY>
__global__ void sgemv_col_major(float *d_A, float *d_x, float *d_y,
const int m, const int n)
{
... Kernel code ...
}
```

- Build switch statements to call the templated function:

```
switch (workSizeX) {
case 0:
      switch (workSizeY) {
      case 0:
            sgemv_col_major<blockSizeX, blockSizeY, 0, 0>
                  <<< dimGrid, dimBlock >>>(d_A, d_temp, m, n); break;
      case 1:
              ...
```

Why ~~How~~ to template the kernel ?

CUDA supports C++ templates (some functionality)

- Template the sgemv kernel as:

```
template <int blockSizeX, int blockSizeY, int workSizeX, int workSizeY>
__global__ void sgemv_col_major(float *d_A, float *d_x, float *d_y,
const int m, const int n)
{
... Kernel code ...
}
```

Resolved at compile time!

```
if (blockSizeY >= 512) {
    if (threadIdx.y < 256) { ... } __syncthreads();
}
```

- Build switch statements to call the templated function:

```
switch (workSizeX) {
case 0:
      switch (workSizeY) {
      case 0:
            sgemv_col_major<blockSizeX, blockSizeY, 0, 0>
                  <<< dimGrid, dimBlock >>>(d_A, d_temp, m, n); break;
      case 1:
              ...
```

Avoiding register spilling

Restictions due to register per block limit

- Blocksize has to be lower than maximum hardware limit

  Tesla C2050: **32768** Cards prior to Fermi: **16384**

- To avoid register spilling, we must limit register usage to

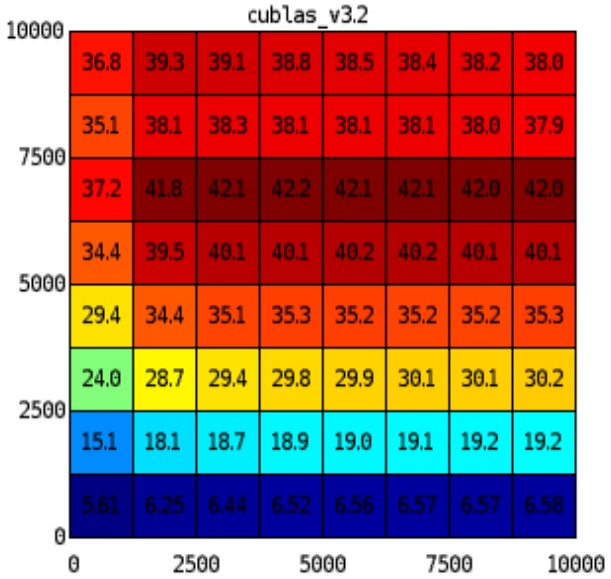
  Register per kernel < `max_registers / blockSize`

by setting the **`--maxrregcount`** compiler option (this is done per .cu file by splitting code up according to blockSize).

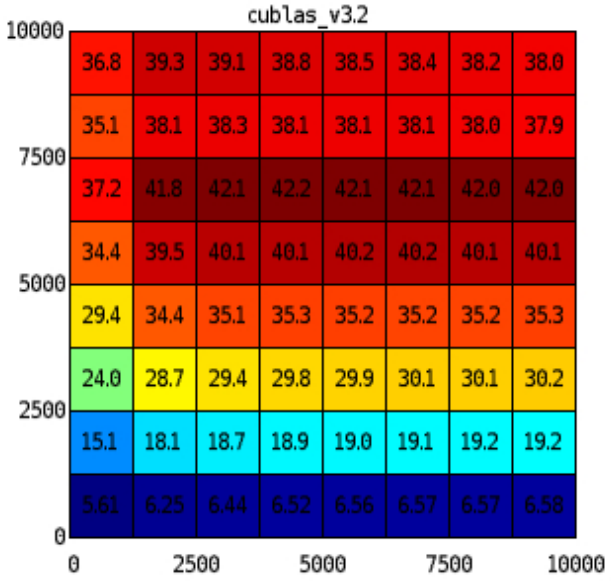
Outline

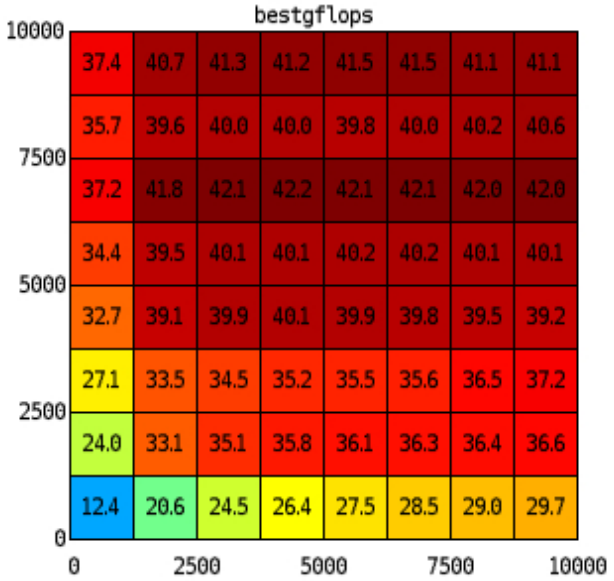
3. Results.

Auto-tuning results [10000x10000]



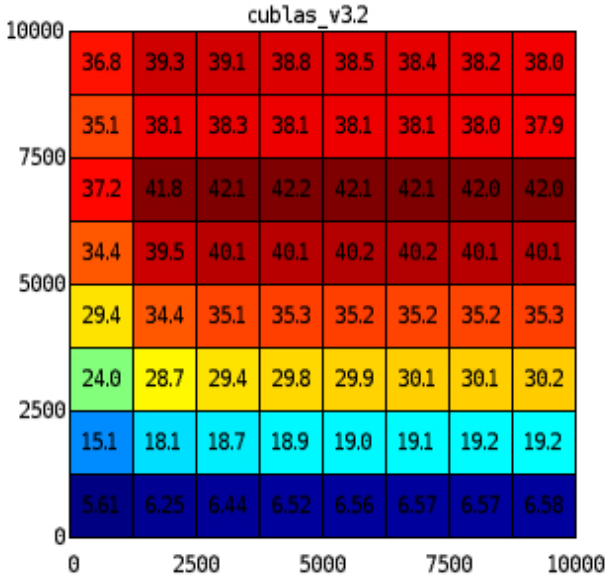
Auto-tuning results [10000x10000]

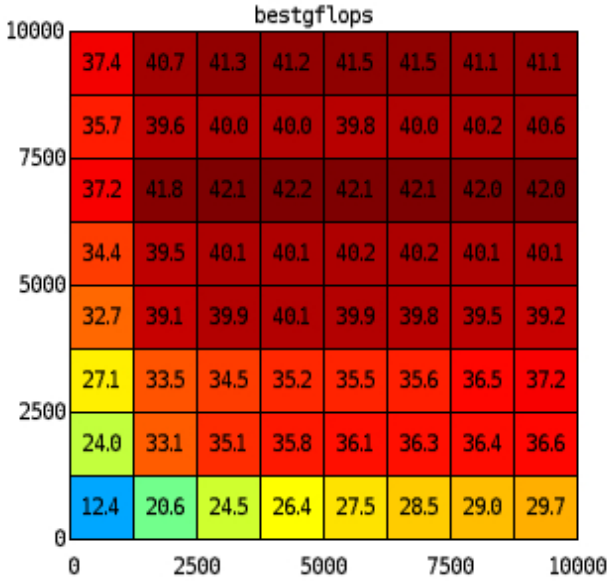


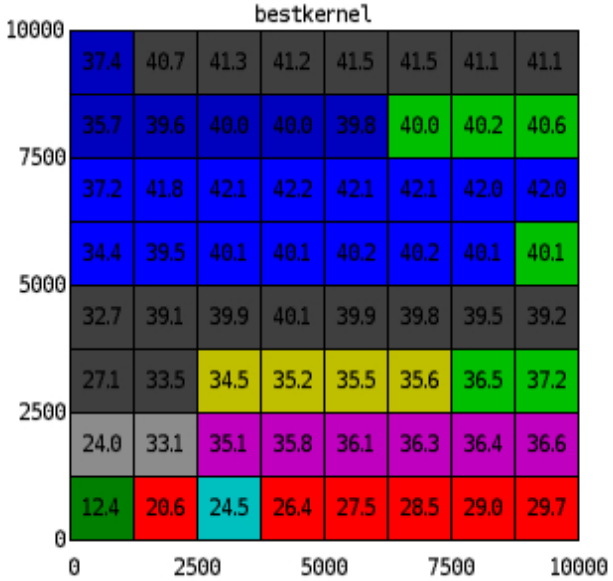


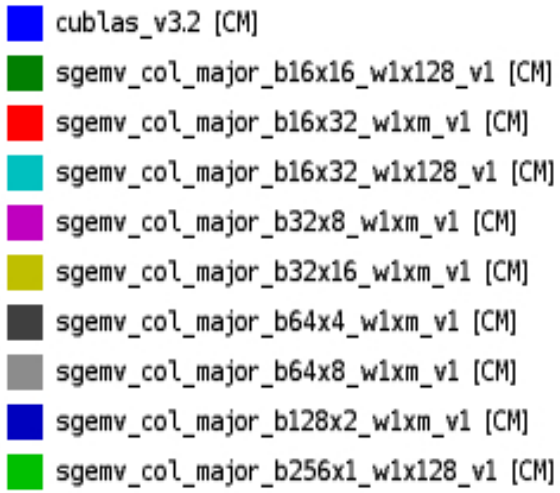
GPU-Lab
DTU Informatics

Auto-tuning results [10000x10000]



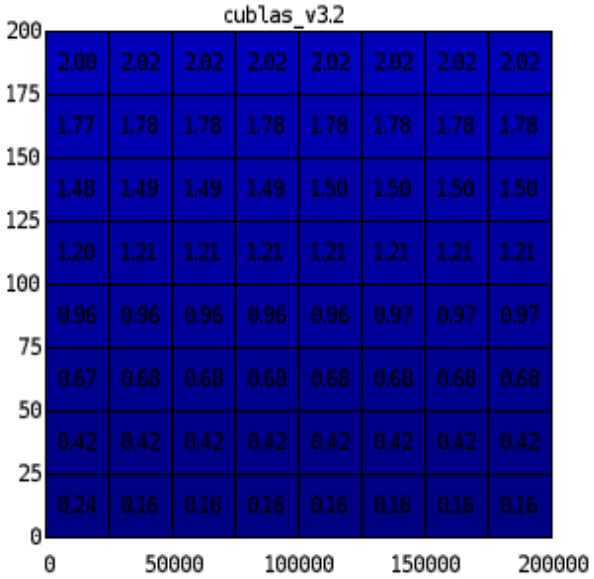


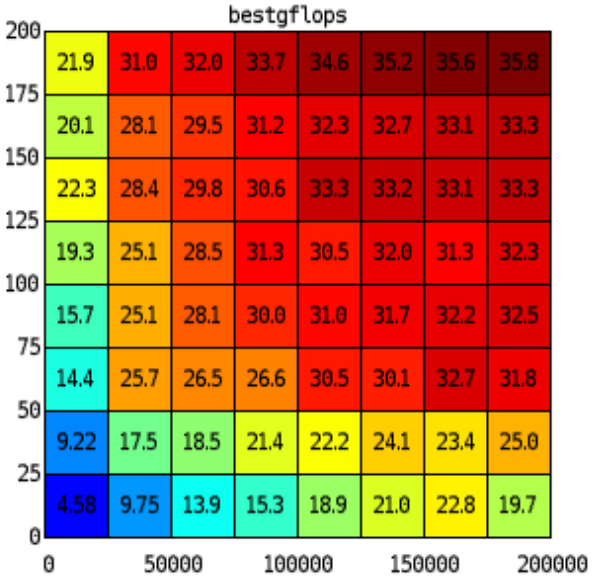


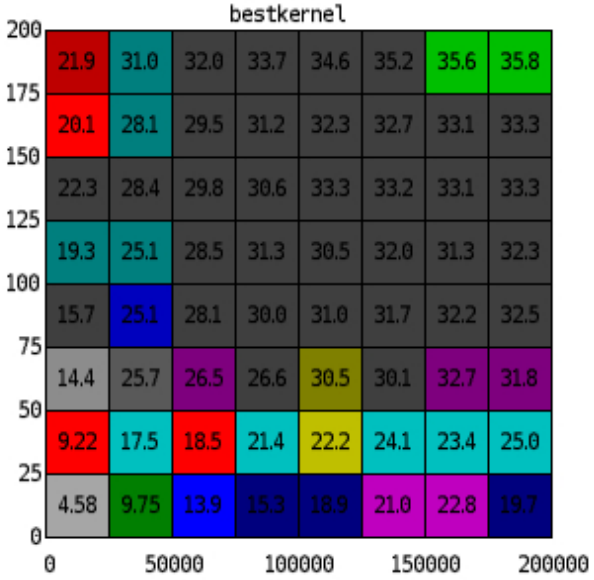


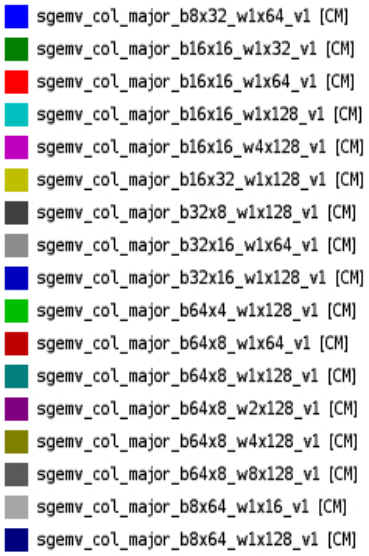
GPU-Lab
DTU Informatics

Auto-tuning results [200x200000]



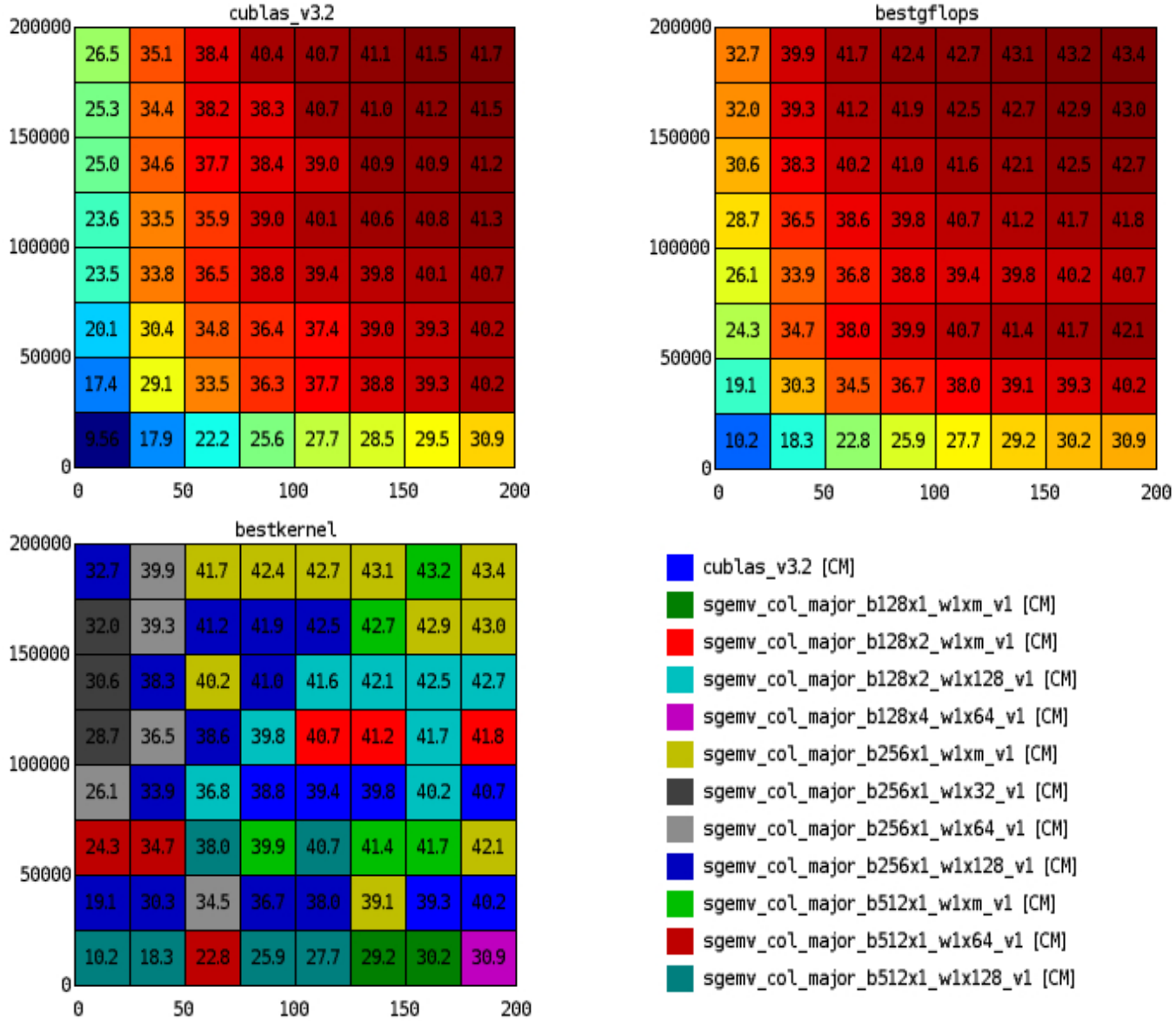






GPU-Lab
DTU Informatics

Auto-tuning results [200000x200]



GPU-Lab
DTU Informatics

Conclusion

We have developed a CUDA auto-tuning framework

- Steps:
  1. Design an intuitively versatile kernel.
  2. Template the kernel with ”performance parameters”.
  3. Build performance model to limit search space.
  4. The auto-tuner selects the best implementation.

Example: Rectangular matrix vector multiplication

- The auto-tuner does a good job (within 1-2 minutes).
- In some cases, we get significantly better performance than the current CUBLAS v3.2 library delivers.